Robust Self-Configuring Embedded Systems
http://www.ece.cmu.edu/roses

Software Architectures for Graceful Degradation in Embedded Systems

Charles Shelton   Philip Koopman

Workshop on Reliability in Embedded Systems
20th Symposium on Reliable Distributed Systems
October 28, 2001

Carnegie Mellon
Electrical & Computer ENGINEERING
Institute for Complex Engineered Systems

Software Architecture for Graceful Degradation

- **Introduction**
  - Software architecture and embedded systems
  - Graceful degradation
  - RoSES product family architecture
- **Example system: an elevator architecture**
  - Elevator Functionality
  - System sensors/actuators
  - Standard elevator architecture
  - Preliminary architecture for graceful degradation
- **Architectural concerns and evaluation**
- **Summary**
- **Future Questions**

Software Architecture for Embedded Systems

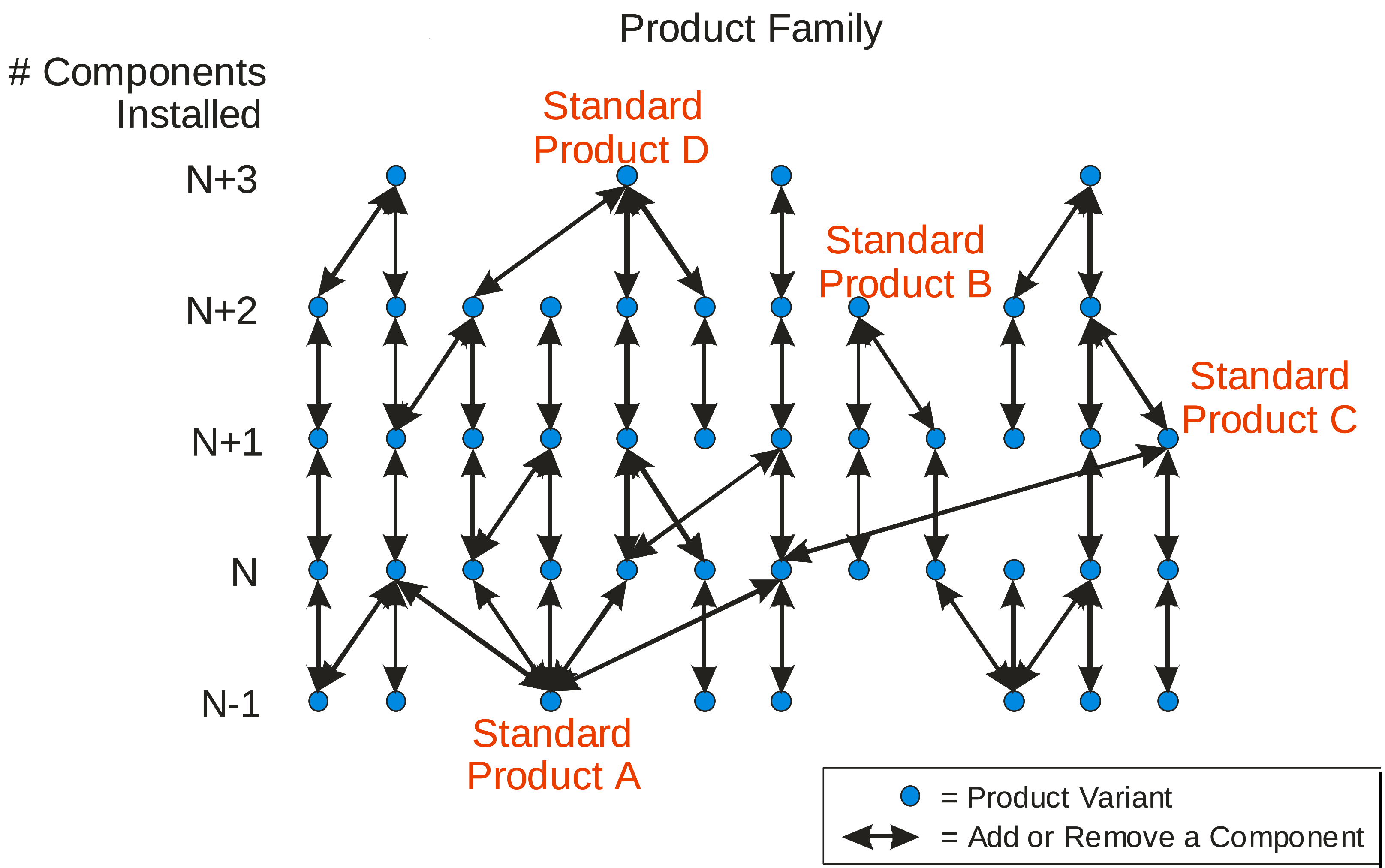
- **Can we develop software architectures to promote graceful degradation in embedded systems?**
- **Software Architecture**
  - Overall structure of system
  - Decompose system into components and connectors
  - Provide ability to reason about system at high level
  - Several architectural styles/patterns have been identified
- **Embedded Systems**
  - Added system complexity/features is driving larger, more complex software
  - Safety-critical, dependability
  - Limited hardware resources, extremely cost-sensitive
  - Traditional software architectural styles may not be appropriate

Graceful Degradation

- **Individual component failures reduce functionality; do not cause system failure**
  - Method to achieve robustness, safety, dependability
- **Goal: Achieve graceful degradation without explicitly specifying all failure scenarios a priori**
  - How can the system's software architecture influence graceful degradation?
- **Possible approaches**
  - Highly distributed
  - No single point of failure
  - Components are decoupled and autonomous
  - Redundancy (not as effective for software)
- **Case Study: Elevator System**

RoSES Product Family Architecture

- Different component configurations provide certain levels of functionality
- Specify architecture with minimum functionality as base configuration
- Focus on architecture for valid component configurations, not reconfiguration problems (Bill Nace's work)

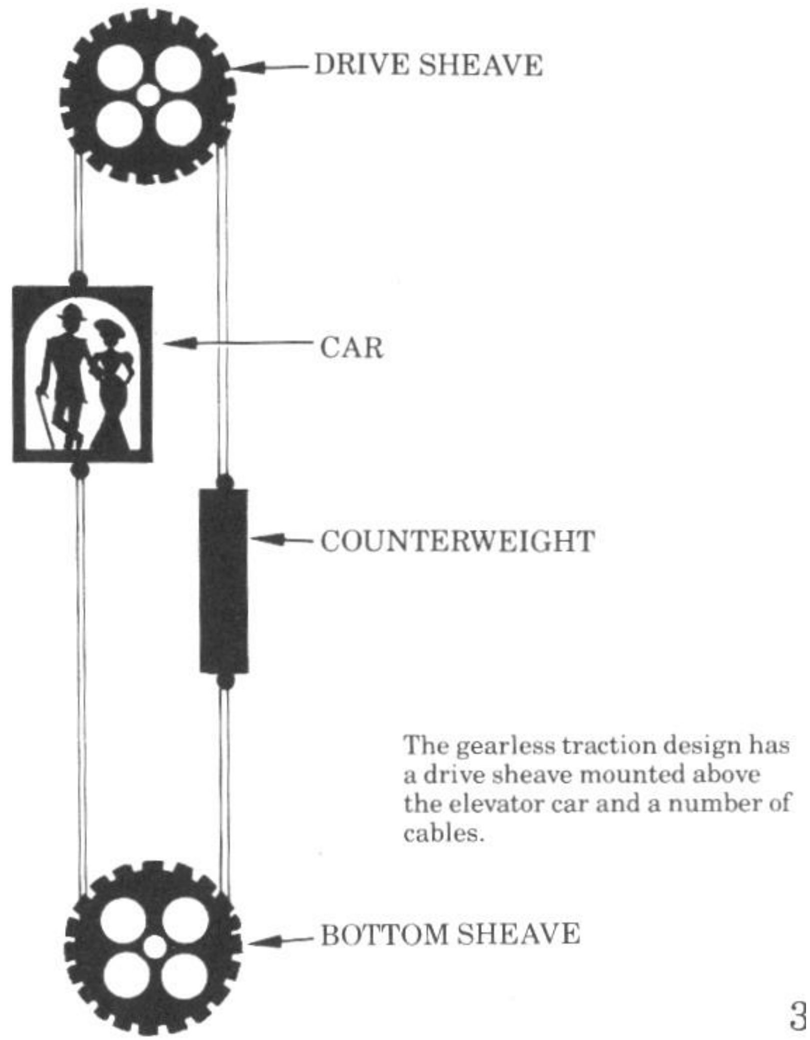


Architectural Decisions

- **Explicitly specify component interfaces**
  - Construct all possible messages to be passed between components
  - Helps determine which components need to communicate
- **Partition Functionality**
  - Separate critical and non-critical functionality
  - Make critical components as autonomous as possible
- **Constrain component configurations**
  - Each component has minimal input/output interface
  - Critical components must be present for base functionality

Elevator Functionality

- **Must transport people between floors**
  - Move car slowly in shaft
  - Stop at every floor
  - Open doors at each floor
- **Must ensure safety**
  - Do not crush people between doors
  - Do not crush people between floor and elevator
  - Do not run car at unsafe speeds
  - Do not trap people in the elevator
- **Optimizations**
  - Only stop on requested floors
  - Provide feedback to passengers
  - Minimize travel time, wait time



Elevator System Sensors and Actuators

Sensors

- Elevator position and speed
  - AtFloor[f,d](v)
  - HoistwayLimit[d](v)
  - DriveSpeed(s,d)
- Door sensors
  - DoorClosed[j](v)
  - DoorOpen[j](v)
  - DoorReversal[j](v)
- Passenger requests
  - CarCall[f](v)
  - HallCall[f,d](b)

Actuators

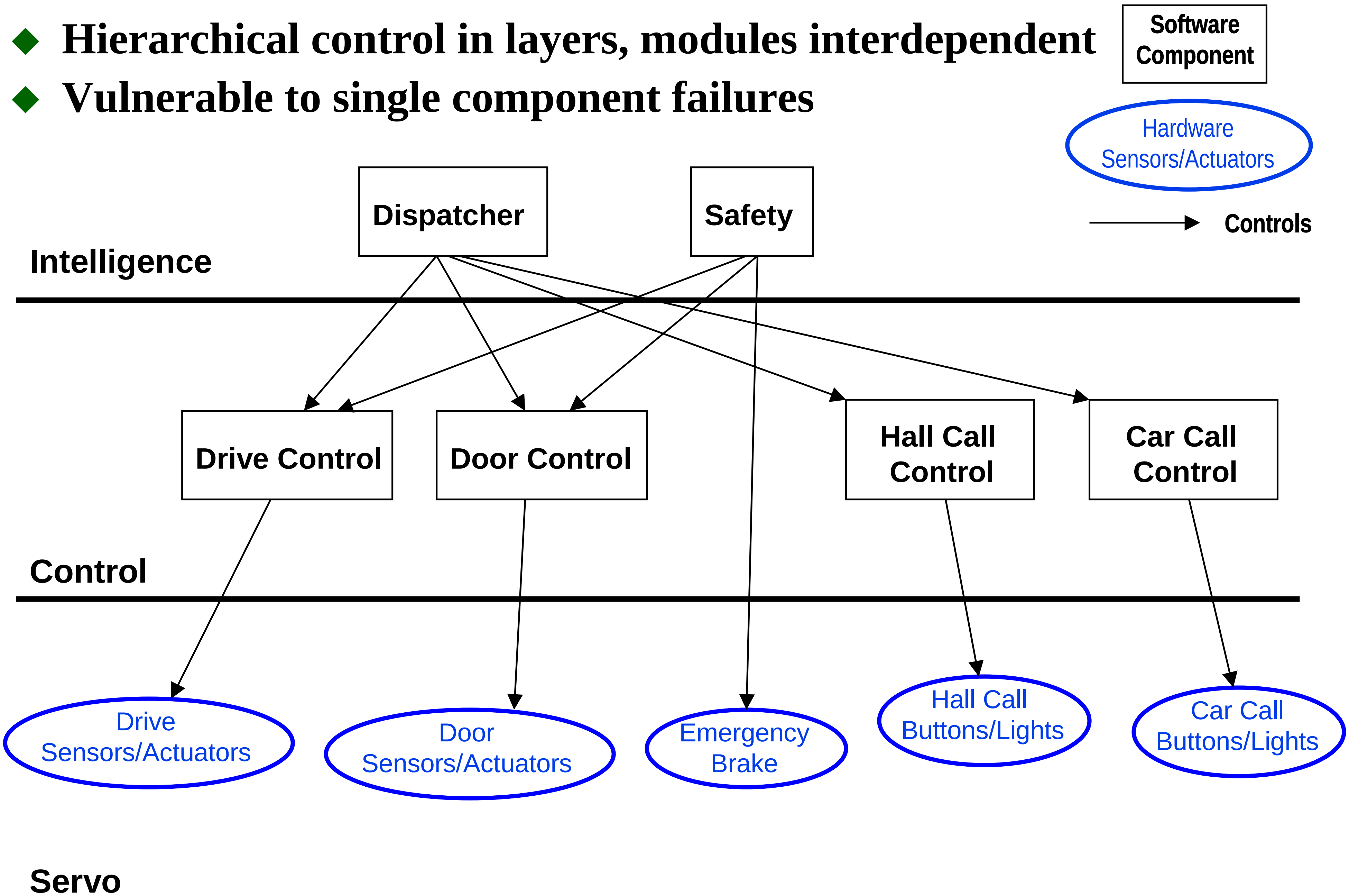
- Elevator control
  - DoorMotor[j](m)
  - Drive(s,d)
  - EmergencyBrake(b)
- Button lights
  - CarLight[f](k)
  - HallLight[f,d](k)
- Passenger feedback
  - CarLantern[d](k)
  - CarPositionIndicator(f)

Control System State

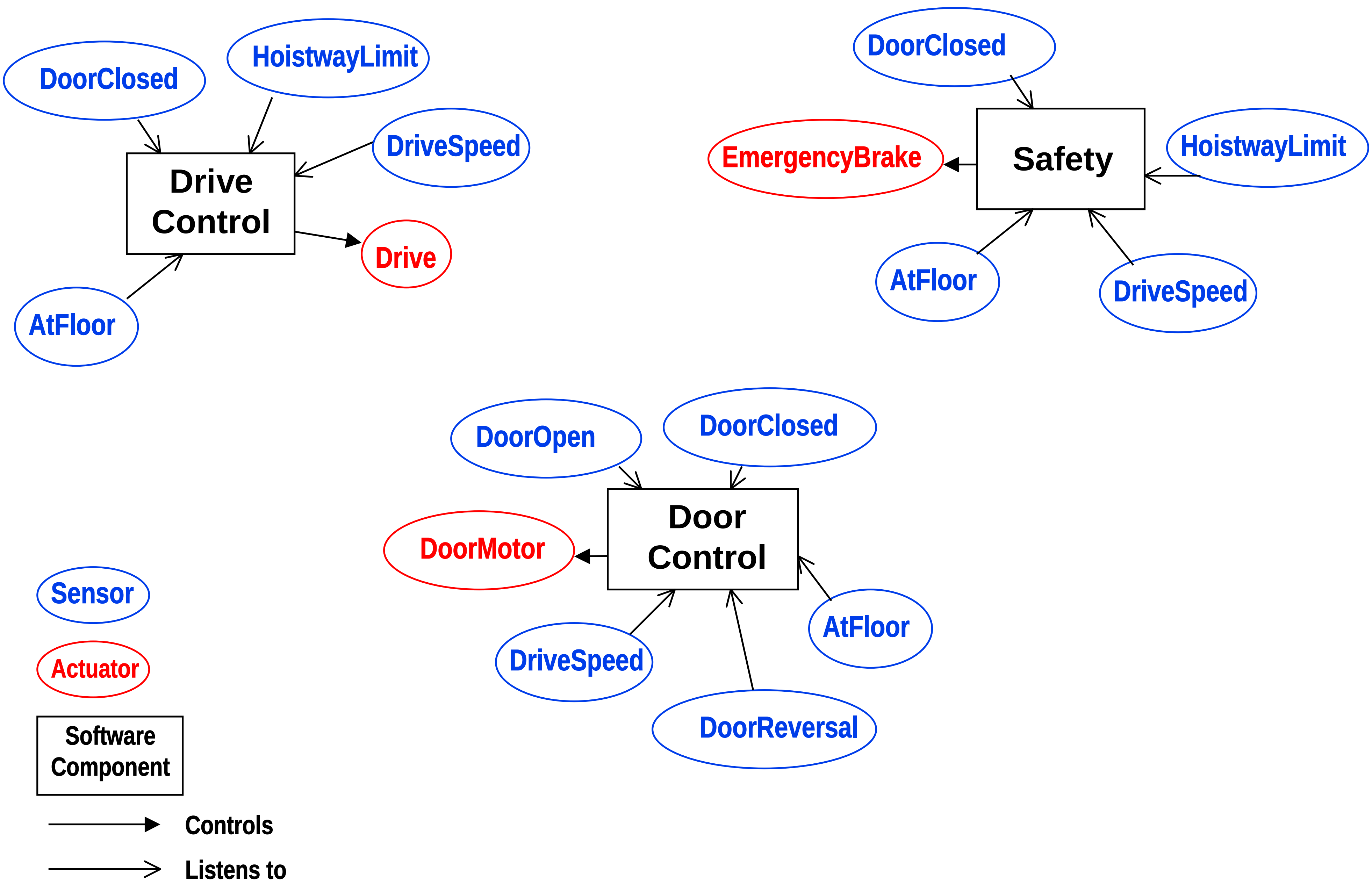
- DesiredFloor(f,d)
- DesiredDwell(n)

Standard Elevator Control Architecture

- Hierarchical control in layers, modules interdependent
- Vulnerable to single component failures

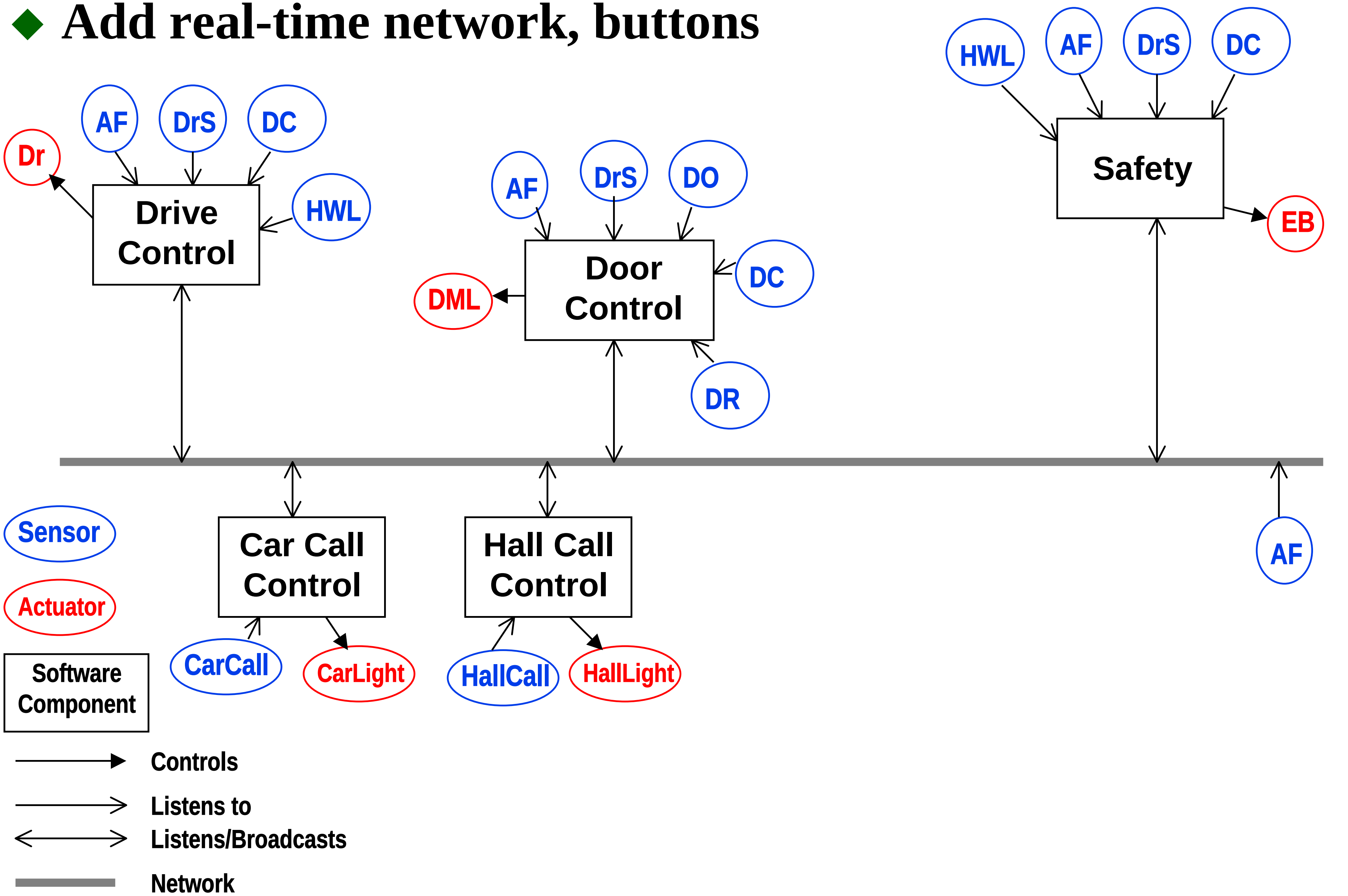


Elevator Architecture: Product 1 (Base)



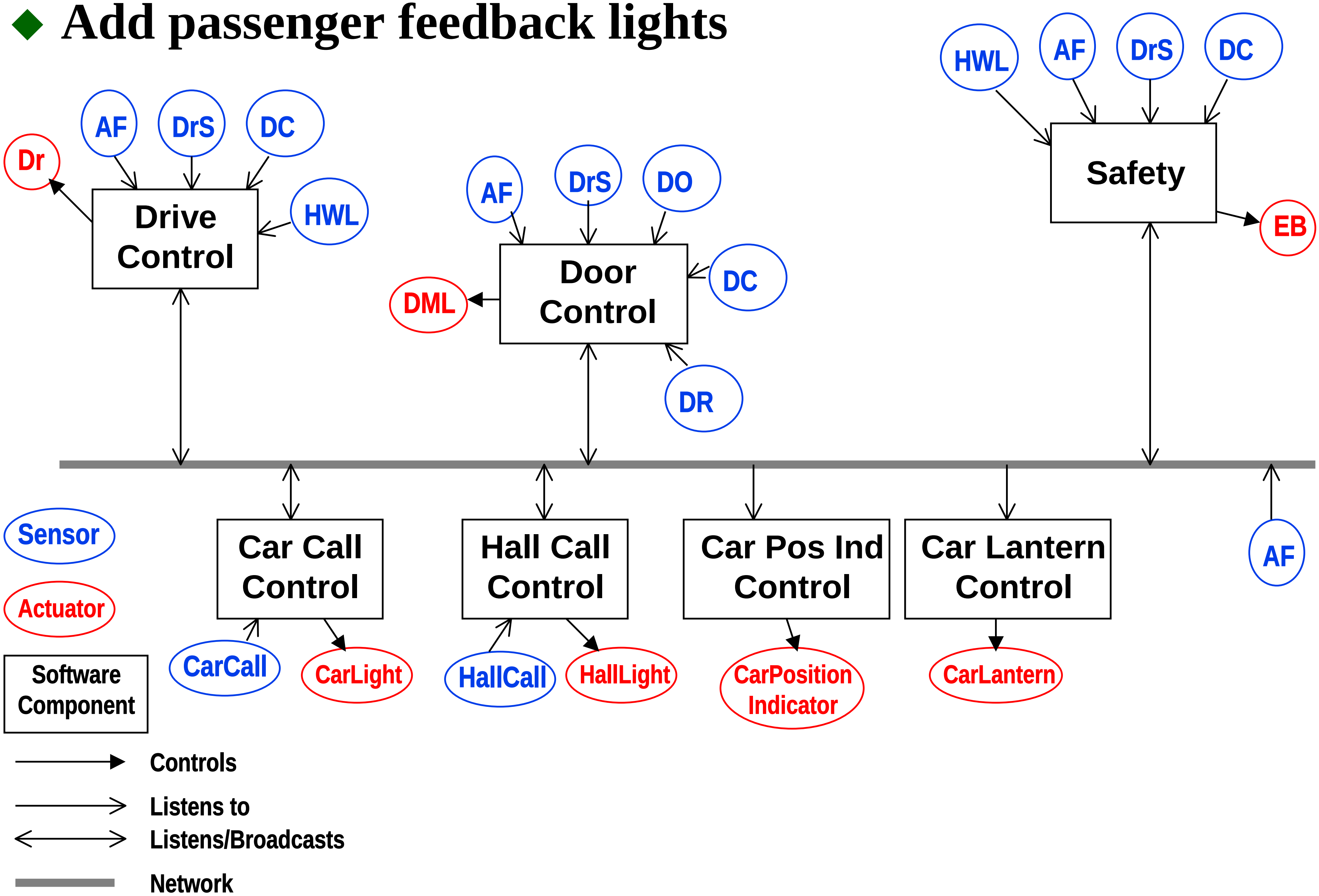
Elevator Architecture: Product 2

- Add real-time network, buttons



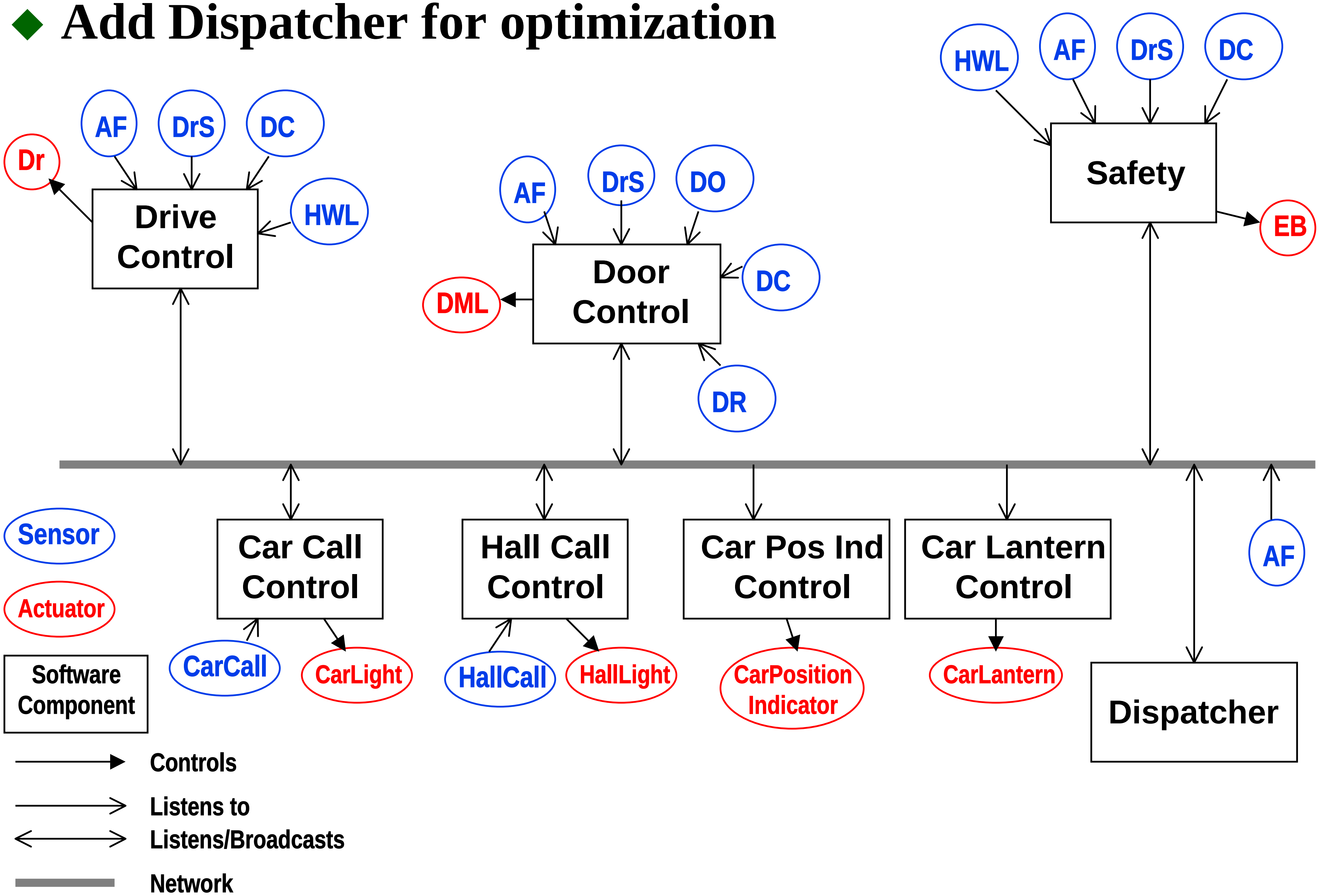
Elevator Architecture: Product 3

- Add passenger feedback lights



Elevator Architecture: Product 4

- Add Dispatcher for optimization



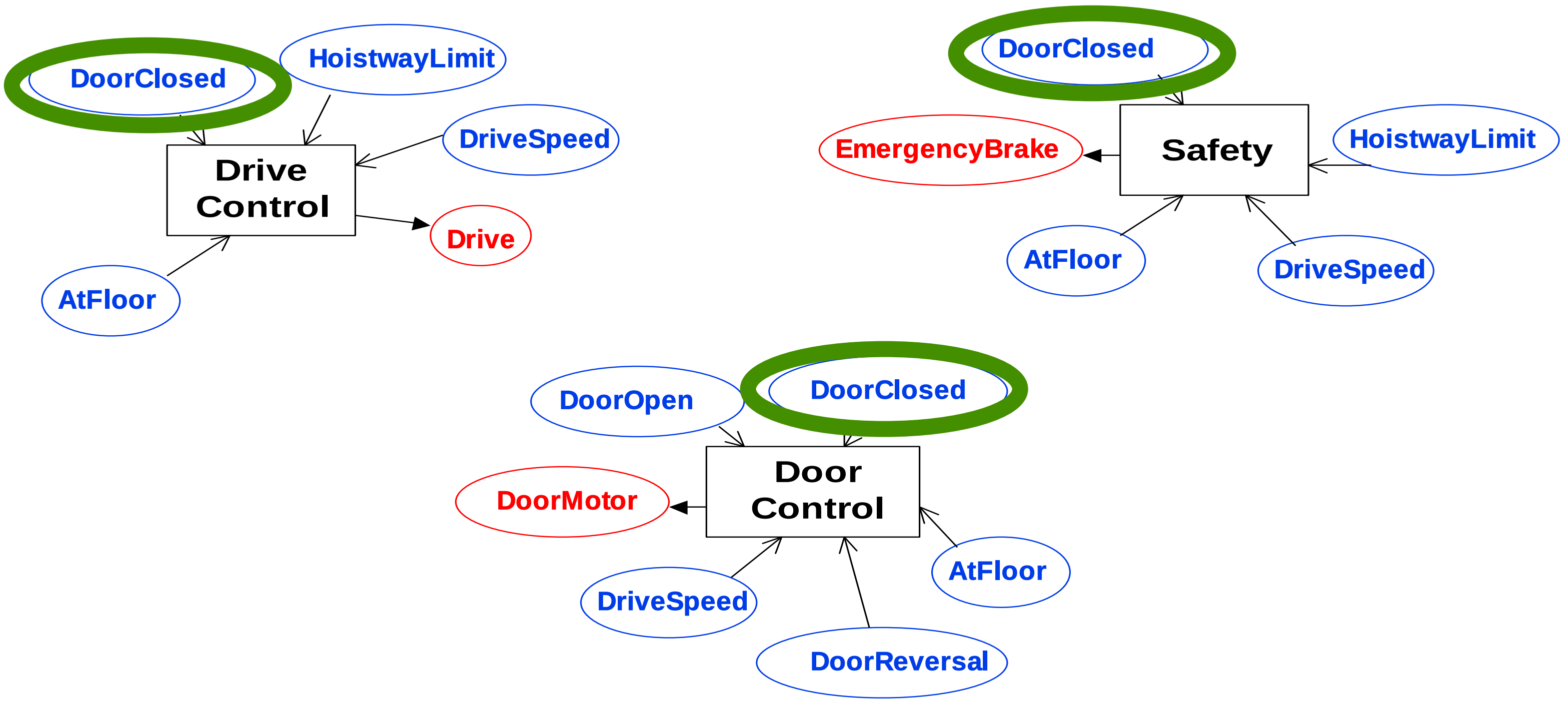
Elevator Control System

- **Main controllers are autonomous**
  - Drive Controller
  - Door Controller
  - Safety
- **Other controllers provide “advisory” information**
  - HallCall buttons
  - CarCall buttons
  - Dispatcher
- **Main controllers follow advice when available**
  - Must pass internal consistency checks
  - In absence of advice, perform base functionality

Architectural Concerns (1)

◆ Cost vs. Safety/Dependability

- Adding additional redundant sensors
  - Necessary to ensure safety for main controllers
  - Could add more for each secondary controller, but cost prohibitive



- Network
  - Could be a single point of failure
  - Without it need exponentially more sensors for more features
  - Could add secondary network to increase dependability

Architectural Concerns (2)

- **Abstract sensor/actuator interface for components**
  - Components can access sensors from physical link or network without modifying code
  - Logical interface separates software concerns from hardware concerns
- **System Configurations**
  - Designed into architecture to constrain configuration options
  - Reconfiguration “hardwired”
  - System should survive components failing in arbitrary order

Evaluation

- **How can I evaluate my architectural design?**
  - Can’t build working elevator and test it
  - Simulation of a distrbuted network
- **Simulation framework exists from ECE 540/549 class**
  - Build executable system from my architecture
  - Fault injection mechanisms to fail components during system operation
  - Measure performance delivering passengers for each configuration

Summary

- **Embedded Systems need methods to ensure safety, dependability, robustness**
  - Graceful Degradation
- **System's software architecture may strongly influence whether graceful degradation is achievable**
- **Design a software architecture for an elevator system**
  - Distributed
  - Decoupling of components
  - Product family structure
  - Some hardware replication
- **Build executable system and test it**
- **How well does it promote graceful degradation?**

Future Questions

- **Can we develop an architectural style specifically for graceful degradation?**
  - Embedded systems have special concerns
    - Cost
    - Constrained resources
- **Can we apply it to multiple domains?**
  - Elevator
  - Automobile navigation system
  - Drive-by-Wire